Matt's Code

Matt James

Contents

Chapter 1

Introduction

This document lists a bunch of the GitHub repositories created by me which may be useful to others. Some of these repositories are fairly complete, others are less so. I will do my best to fix and update anything that is buggy or incomplete, please do report bugs in the relevant repositories if you can. If you're feeling particularly helpful - feel free to send pull requests.

Most of the code here is written in Python, some things make use of C++ libraries to do some of the heavy lifting, one is a pretty dodgy Python wrapper of a C++ wrapper of Fortran code... Some of the modules and libraries used here are dependencies of others. In the more complete repos `pip` will take care of dependencies, otherwise some manual installation may be required.

1.1 Setting up the environment

In this section I describe how to set up the environment such that everything *should* pretty much work...

1.1.1 Linux

If running on ALICE/SPECTRE, you will most likely be required to enable the following modules:

```
module load gcc/9.3
module load python/gcc/3.9.10
module load git/2.35.2
```

where exact version numbers may change (use whatever is latest, don't just copy and paste!) and the replacement for SPECTRE/ALICE may have another method for loading these things in for all I know. I also recommend adding those to the end of your `/.bashrc` file so that they load every login, e.g.:

```
echo module load gcc/9.3 >> ~/.bashrc
echo module load python/gcc/3.9.10 >> ~/.bashrc
echo module load git/2.35.2 >> ~/.bashrc
```

The above is unlikely to be necessary on a local Linux installation, instead I would recommend installing `git, gcc, g++, make, gfortran` and `pip3`, e.g. in Ubuntu:

```
sudo apt install git gcc g++ binutils gfortran python3-pip
```

All of the above should allow you to install/run/compile most of my code. I wouldn't recommend using Conda in Linux - I know it has cause some problems/confusion when it comes to linking Python with C/C++ on SPECTRE.

1.1.2 Windows

A fair portion of the code is able to run on Windows - much of the Python code is platform independent and some of the C++ libraries/backends are able to be compiled using Windows. In this case, I *would* actually recommend installing Conda, as it worked for me. The GCC compilers (for C/C++/Fortran) can all be installed easily with TDM-GCC (get the 64-bit version here), just remember to put a tick in the box for "fortran" and "openmp".

1.1.3 MacOS

I managed to install the relevant packages in a virtual Hackintosh once. I don't remember how, perhaps using homebrew. Good luck...

1.2 Setting up a virtual environment

In SPECTRE I never actually bothered with a virtual environment, mistakes were made, headaches may have been avoided had I done so. This step is entirely optional, but somewhat recommended:

```
#create a virtual environment, call it what you want,
#here I call mine "env"
python3 -m venv env
```

Once this has been created, you MUST activate it before running any code, or you will just be running things globally:

```
source env/bin/activate
```

note that I am assuming that `env` exists in the current working directory, if not adjust the path accordingly! If it works, the prompt terminal prmpt should change, e.g:

```
#before:
matt@matt-MS-7B86:~$

source env/bin/activate
#after:
(env) matt@matt-MS-7B86:~$
```

1.3 Some python packages

Here are a list of Python packages which are either going to be required by most of my code, or would just be recommended:

1. ipython : best Python interpreter, forget notebooks
2. numpy : essential, don't skip
3. matplotlib : for plotting
4. scipy : loads of good stuff here
5. wheel : used to build Python packages to be installed by pip
6. cdflib : reads CDF files
7. keras : nice for machine learning
8. tensorflow : also machine learning

install them:

```
#update pip first
python3 -m install pip --upgrade --user

pip3 install ipython numpy matplotlib scipy wheel cdflib keras tensorflow --user
```

where the “‘`--user`”’ flag may or may not be necessary, depending on your version of Python - it places the installed modules in ~`/.local/lib/python3.9/site-packages`.

In theory, at this point you should be able to run `ipython3` (or just `ipython`) within the terminal, from which any installed code can be imported. The reason I recommend using Ipython over the standard Python interpreter is that it has autocomplete and it uses pretty colours for syntax highlighting. It would also be a good idea to enable the autoreload feature in Ipython, which recompiles anything that has been edited since it was last run, otherwise would have to reload the code manually (or restart the session) after every edit. Run

```
ipython profile create
```

then add the following lines to ~`/.ipython/profile_default/ipython_config.py`:

```
c.InteractiveShellApp.extensions = ['autoreload']
c.InteractiveShellApp.exec_lines = ['%autoreload 2']
c.InteractiveShellApp.exec_lines.append('print("Warning: disable autoreload in ipython_config.p
```

That should just about do it.

Chapter 2

Plasma Models

2.1 `spicedmodel`: The Scalable Plasma Ion Composition and Electron Density Model

2.2 `HermeanFLRModel`: Model of Mercury's dayside plasma mass density

2.3 `PyGCPM`: Wrapper for the Global Core Plasma Model

Chapter 3

Field Models

3.1 `PyGeopack`: Python wrapper for the Tsyganenko field models

3.2 `geopack`: C++ wrapper for Tsyganenko field models

3.3 `libinternalfield`: C++ spherical harmonic model code

3.4 `vsmodel`: Python based Volland-Stern electric field model for Earth

3.5 `JupiterMag`: Python wrapper for Jovian field models

3.6 `libjupitermag`: C++ library for field tracing in Jupiter's magnetosphere

3.7 `con2020`: Python implementation of Jupiter's magnetodisc model

3.8 `libcon2020`: C++ implementation of Jupiter's magnetodisc model

3.9 `jrm33`: The JRM33 model in Python

3.10 `jrm09`: The JRM09 model in Python

3.11 `vip4model`: The VIP4 model in Python

Chapter 4

Spacecraft Data

4.1 `Arase`: Download and read Arase data

4.2 `RBSP`: Download and read Van Allen Probe data

4.3 `cluster`: Download and read Cluster data

4.4 `pyCRRES`: Download and read CRRES data

4.5 `themissc`: Download and read THEMIS data

4.6 `imageeuv`: Download and read IMAGE EUV data

4.7 `imagerpi`: Download and read IMAGE RPI data

4.8 `imagePP`: Download and read Goldstein's plasmapause dataset

4.9 `PyMess`: Download and read MESSENGER data

4.10 `FIPSProtonData`: Download and read ANN verified FIPS moments

4.11 `VenusExpress`: Download and read VEX data

Chapter 5

Ground Data and Geomagnetic Indices

5.1 groundmag: Tools for processing and reading ground magnetometer data

5.2 SuperDARN: Simple SuperDARN fitacf reading code

https://github.com/mattkjames7/SuperDARN

The SuperDARN module is for reading and plotting SuperDARN fitacf files. It is a fairly simple tool, but use with caution because there may be some errors...

5.2.1 Installation

This package is not in the PyPI, so manual installation is necessary:

```
#clone the repo
git clone https://github.com/mattkjames7/SuperDARN
cd SuperDARN

#build a Python package
python3 setup.py bdist_wheel

#install it (replace 0.1.0 with whatever version is built)
pip3 install dist/SuperDARN-0.1.0-py3-none-any.whl --user
```

Once installed, the directory used to create the Python wheel file can be deleted. It can be uninstalled using `pip3 uninstall SuperDARN`.

Before running for the first time, a couple of environment variables need to be set up to tell the module where to look for fitacf files and to say where it is able to store some files:

```
#path to where FITACF files are stored
#(this one is specific to SPECTRE)
```

```
export FITACF_PATH=/data/sol-ionosphere/fitacf

#path to where this module can create some files
#(this should be a path where you have write access)
export SUPERDARN_PATH=/some/other/path/SuperDARN
```

This module will not currently run on Windows (as far as I am aware) because it requires the compilation of some C++ code which is not yet cross-platform.

Usage

In ipython, the first time this module is imported, it should attempt to download some files from the Radar Software Toolkit (RST) which help in calculating the coordinates of the fields of view of each radar. These files are created in the path defined be the `$SUPERDARN_PATH` variable.

Reading Data

There are a few functions within `SuperDARN.Data` which provide objects containing data:

```
import SuperDARN as sd

#get the data from a single cell (Radar,Date,ut,Beam,Gate)
cdata = sd.Data.GetCellData('han',20020321,[22.0,24.0],9,25)

#or a whole beam of data (Radar,Date,ut,Beam)
bdata = sd.Data.GetBeamData('han',[20020321,20020322],[22.0,24.0],7)

#data for the whole field of view (Radar,Date,ut)
#in this case, the output is a dict where each key is a beam number
#pointing to a recarray for each beam as produced by GetBeamData
rdata = sd.Data.GetRadarData('han',[20020321,20020322],[22.0,23.0])
```

In the above examples `bdata` and `cdata` are `numpy.recarray` objects, `rdata` is a `dict` object containing a `numpy.recarray` for each beam.

The fitacf data are stored in memory once loaded so that they don't need to be re-read every time the data are requested. To check how much memory is in use and to clear it:

```
#check memory usage in MB
sd.Data.MemUsage()

#clear memory
sd.Data.ClearData()
```

Plotting Data

There are a bunch of very simple plotting functions, e.g.:

```
import matplotlib.pyplot as plt

#create a figure
plt.figure(figsize=(8,11))

#plot the power along a beam
ax0 = sd.Plot.RTIBeam('han',[20020321,20020322],[23.0,1.0],9,[20,35],
                                        Param='P_l',ShowScatter=True,fig=plt,
                                        maps=[2,3,0,0],scale=[1.0,100.0],zlog=True,
                                        cmap='gnuplot')

#the velocity
ax1 = sd.Plot.RTIBeam('han',[20020321,20020322],[23.0,1.0],9,[20,35],Param='V',
                                        fig=plt,maps=[2,3,1,0])

#velocity along a range of latitudes at a ~constant longitude of 105
ax2 = sd.Plot.RTILat('han',[20020321,20020322],[23.0,1.0],105.0,Param='V',
                                        fig=plt,maps=[2,3,0,1])

#velocity along a range of longitudes at a ~constant latitude of ~70
ax3 = sd.Plot.RTILon('han',[20020321,20020322],[23.0,1.0],70.0,Param='V',
                                        fig=plt,maps=[2,3,1,1])

#some specific cells
beams = [1,5,7,2,8,4,9]
gates = [20,26,33,22,25,21,29]
ax4 = sd.Plot.RTI('han',[20020321,20020322],[23.0,1.0],beams,gates,
                                        Param='V',fig=plt,maps=[2,3,0,2])

#totally different FOV plot
ax5 = sd.Plot.FOVData('han',20020321,23.5,Param='V',fig=plt,maps=[2,3,1,2])


plt.tight_layout()
```

which should produce figure 5.1.

Fields of View

These may be wrong. Use with great caution.

The fields of view of each radar are stored as instances of the `SuperDARN.FOV.FOVObj` objects in memory and can be accessed using `GetFOV`, e.g.:

```
#get the object from memory
Date = 20020321
fov = sd.FOV.GetFOV('pyk',Date)

#use it to retrieve the FOV in mag coordinates
mlon,mlat = fov.GetFOV(Mag=True,Date=Date)

#plot it
```

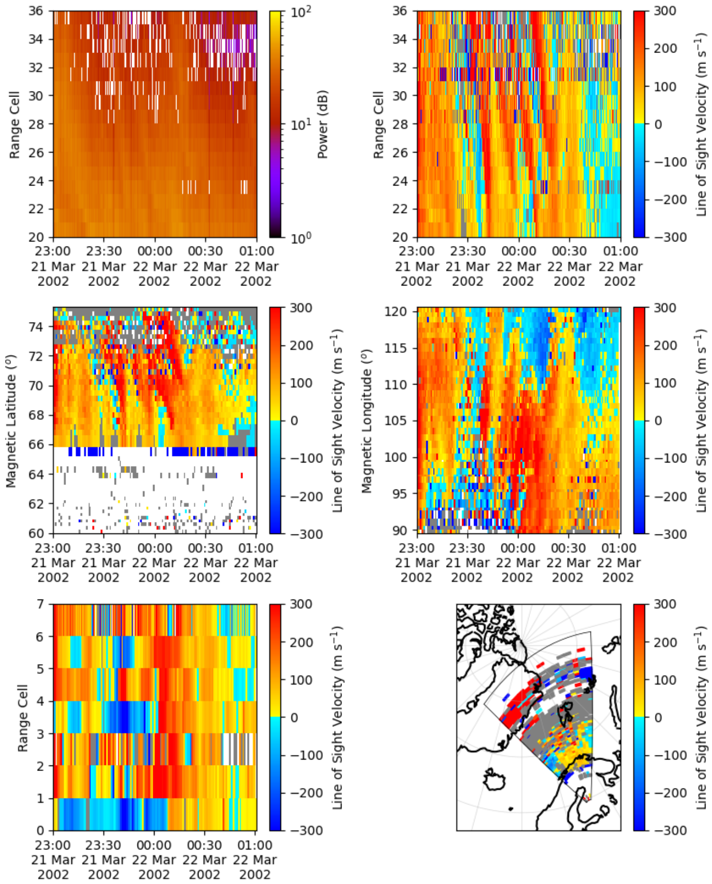

Figure 5.1: Top left: range time intensity (RTI) plot of backscatter power. Top right: RTI plot of line of sight velocity. Mid left: velocity along a line of cells in magnetic longitude. Mid right: velocity along a range of longitudes. Bottom left: velocity of specific range cells. Bottom right: velocity within the field of view plot.

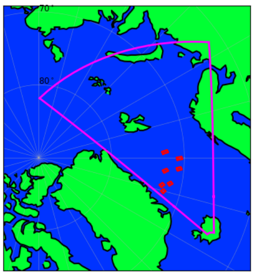

Figure 5.2: SuperDARN field of view plot with specific cells highlighted.

```
ax = fov.PlotPolar(Background=[0.0,0.2,1.0],Continents=[0.0,1.0,0.2],
                                color='magenta',ShowBeams=False,ShowCells=False,
                                linewidth=2.0,Mag=True,Lon=True)

#add some cells
beams = [1,5,7,2,8,4,9]
gates = [20,26,33,22,25,21,29]
fov.PlotPolarCells(beams,gates,color='red',fig=ax,Mag=True,linewidth=2.0,Lon=True)
```

The above code should look like figure 5.2:

5.3 kpindex: Download the latext Kp indices

5.4 pyomnidata: Download the latext OMNI and solar flux data

5.5 smindex: Read the SuperMAG indices

Chapter 6

Machine Learning

6.1 `NNClass`: Simple neural network classifier module

6.2 `NNFunction`: Train neural networks on arbitrary functions

Chapter 7

Other Tools

7.1 `wavespec`: Spectral analysis tools

7.2 `MHDWaveHarmonics`: Tools for MHD waves

7.3 `FieldTracing`: Python field tracing code

7.4 `DateTimeTools`: Tools for dealing with dates and times

7.5 `datetime`: C++ library dealing for dates and times

7.6 `PyFileIO`: Tools for reading and writing files

7.7 `RecarrayTools`: Tools for manipulating `numpy.recarrays`

7.8 `PBSJobExamples`: Examples for submitting jobs to PBS

7.9 `PlanetSpice`: SPICE related code

7.10 `ColorString`: Change colour of strings in the terminal

7.11 `cppembedbinary`: Examples for embedding data into C++ code

7.12 `libspline`: C++ library for splines

7.13 `linterp`: C++ interpolation code